Team 16

# Voting System

Software Design Document

Names:
Crystal Wen
Shunichi Sawamura
Lysong Seang
Fumisato Teranishi

Date: (02/15/2024)

## TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 Purpose

This software design document describes the architecture and system design of the Voting System. This document is intended to be read by people who will be using, testing, and modifying the system of the Voting System. This includes election officials who will use the Voting System to allocate seats and determine the winner(s) of the election, testers who will test for any bugs in the system, and the programmers who will develop and maintain the Voting System.

## 1.2 Scope

This stand-alone Voting System is designed to handle the ballot counting according to either Open-Party Listing (OPL) or Closed-Party Listing (CPL). Overall, when the program receives a formatted file containing the election information, this program is designed to process the ballots, determine a winner, and allocate seats to each party. Additionally, it will provide an audit file and display the election information.

The goal of the Voting System is to efficiently and quickly count the ballots according to the election type and accurately allocate seats using the Largest Remainder Method. The objective of this system is to effectively reduce the amount of time it takes to process ballot counts which will allow the election officials to promptly receive the results and statistics of the election.

## 1.3 Overview

Chapter 2 gives an overview of the system and its organization. It gives a general explanation of the main components and how the system behaves in context.

Chapter 3 provides the architectural design that specifies the subsystems that perform all of the functions within the Voting System. It also contains high-level diagrams decomposing each function to its low-level description. This includes a UML activity diagram, a UML class diagram, and a sequence diagram.

Chapter 4 discusses the data structure design.

Chapter 5 includes the component designs of the functions used in the Voting System.

Chapter 6 discusses the user interface design and the screen objects that will be used in the design.

Chapter 7 contains a requirement matrix to show which system components satisfy each of the functional requirements from the SRS.

### 1.4 Reference Material

System Design Document: https://canvas.umn.edu/courses/413086/files/41638355?wrap=1

UML Diagrams: https://canvas.umn.edu/courses/413086/files/41477667?wrap=1

### 1.5 Definitions and Acronyms

**CSV:** Comma Separated Values, a type of file that uses commas to separate data.

**OPL:** Open Party Listing

**CPL:** Closed Party Listing

**SDD**: Software Design Document

**SRS:** Software Requirements System

**UML:** Unified Modeling Language

## 2. System Overview

The Voting System is a stand-alone product designed to count ballots, allocate seats, and determine the winners and winning parties. The Voting System is designed to process ballots from two types of elections, OPL and CPL. When a CSV file is provided, the system will use the information provided in the file to count the votes each party and candidate received, determine the winner, and allocate seats to each party. Lastly, an audit file will be produced with all of the election information, and the results and statistics of the election will be displayed on screen.

## 3. System Architecture

### 3.1 Architectural Design

The UML class diagram is shown in Figure 3.1. It can also be referenced at *UMLClassDiagram_team16.pdf* with higher resolution. It describes the structure of our voting system with the relationships between classes. The data structure of class attributes and methods are described in Section 4, and the methods are described in detail in Section 5.

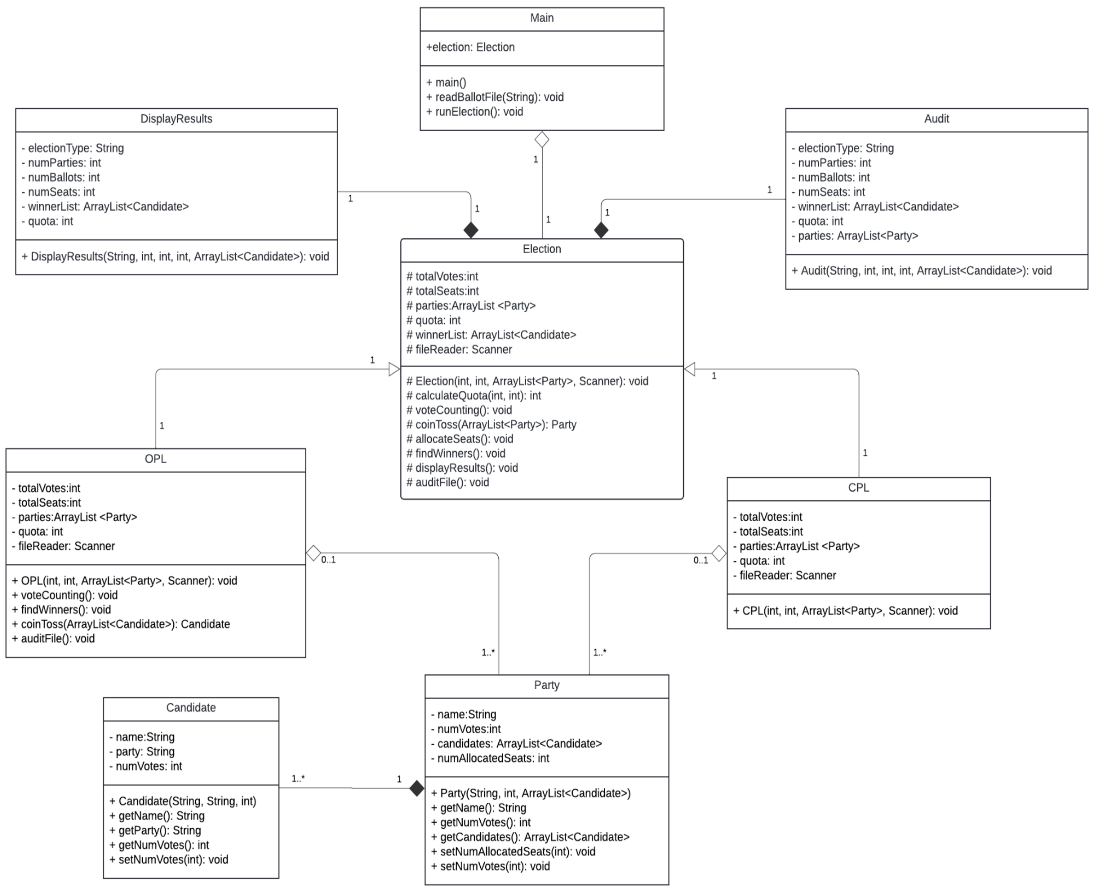


**Figure 3.1 UML Class Diagram**

## 3.2 Decomposition Description

The activity diagram for the CPL is shown in Figure 3.2.1. It can also be referenced at *ActivityDiagramCPL_team16.pdf* with higher resolution. The activity diagram starts with the election date being set, and it goes through the process of the election until all votes have been sent in. After all of the ballots have been collected and compiled in a CSV file, an election official or other users will open it in the Voting System. It will be determined, in the file, that the election was a CPL-type election and begins the vote counting and seat allocation process. Once all votes have been counted and seats have been filled, a winner will be determined by the number of votes. Should there be a tie, a fair coin toss will determine the winner.

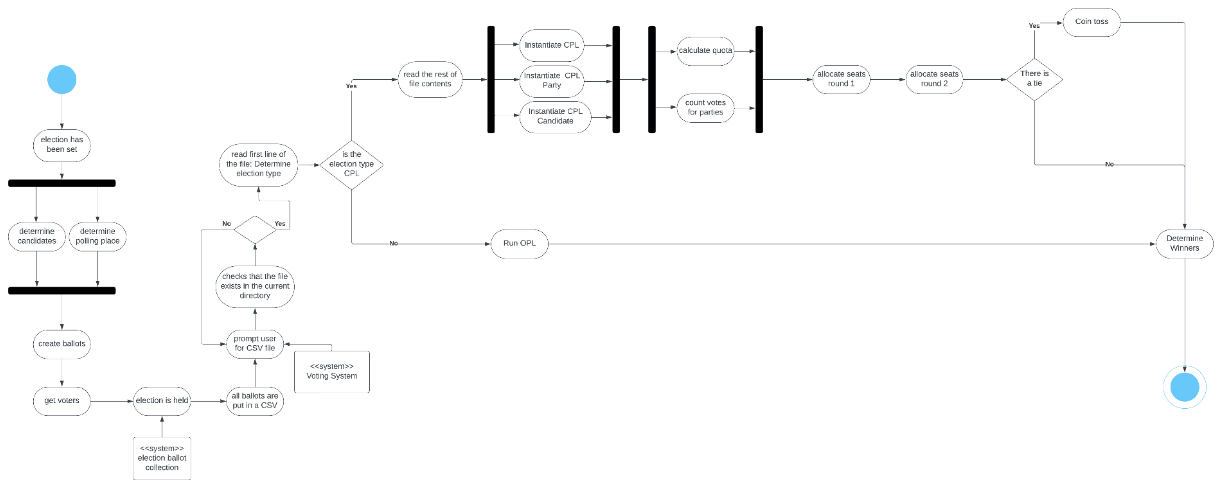


**Figure 3.2.1 UML Activity Diagram for the CPL election.**

The sequence diagram for the running of the open party listing (OPL) election is shown in Figure 3.2.2. It also can be referenced at *SequenceDiagramOPL_team16.pdf* with higher resolution. The system always starts from the Main(). Once the program runs, it provides the text prompt to the screen to get the CSV filename and reads the file contents through readBallotFile(). The readBallotFile() method will only fetch the information about the election type, number of seats, number of ballots, number of candidates, candidates, and their parties. After that, Main uses the runElection() method to analyze the ballot results. This method first instantiates Election, Party, and Candidate objects based on the collected information in the readBallotFile(). There are two types of Election classes: CPL and OPL. While their constructor receives the same input, they have different algorithms to determine winners. Therefore, the OPL class overrides some methods from its parent Election class. After the system successfully instantiates the OPL/CPL, Party, Candidate class, it runs the voteCounting() to read the rest of the file contents and clarify how many votes each party or candidate obtains. In CPL, the count is incremented with respect to each party, although in OPL, the count is incremented with respect to each candidate and the party is the sum of all the votes the affiliated candidates received. When it finishes counting, the number of votes obtained is stored using the setNumVotes() method in Party and Candidate. The system also calculates the quota value in the calculateQuota(). Then, the system will begin to allocate seats with the allocateSeats() method. The method will allocate seats by dividing each party's votes by the quota and each party will receive a number of seats equal to that quotient. If there are any remaining seats, they will be allocated to the parties in a round-robin fashion starting with the party with the largest remainder. If it is CPL, the order of candidates in the party is predetermined. If it is OPL, the order of candidates in the party is determined by the number of votes they received. If there is a tie during the seat allocation, we will use the coinToss() method to decide a winner. Once the system completes the seat allocation, it clarifies winners with findWinners() method and stores winner information in the winnerList field. The audit file will be generated in the auditFile() method and Audit class to store election results. The election results also will be displayed on the screen through the displayResults() method and DisplayResults class.

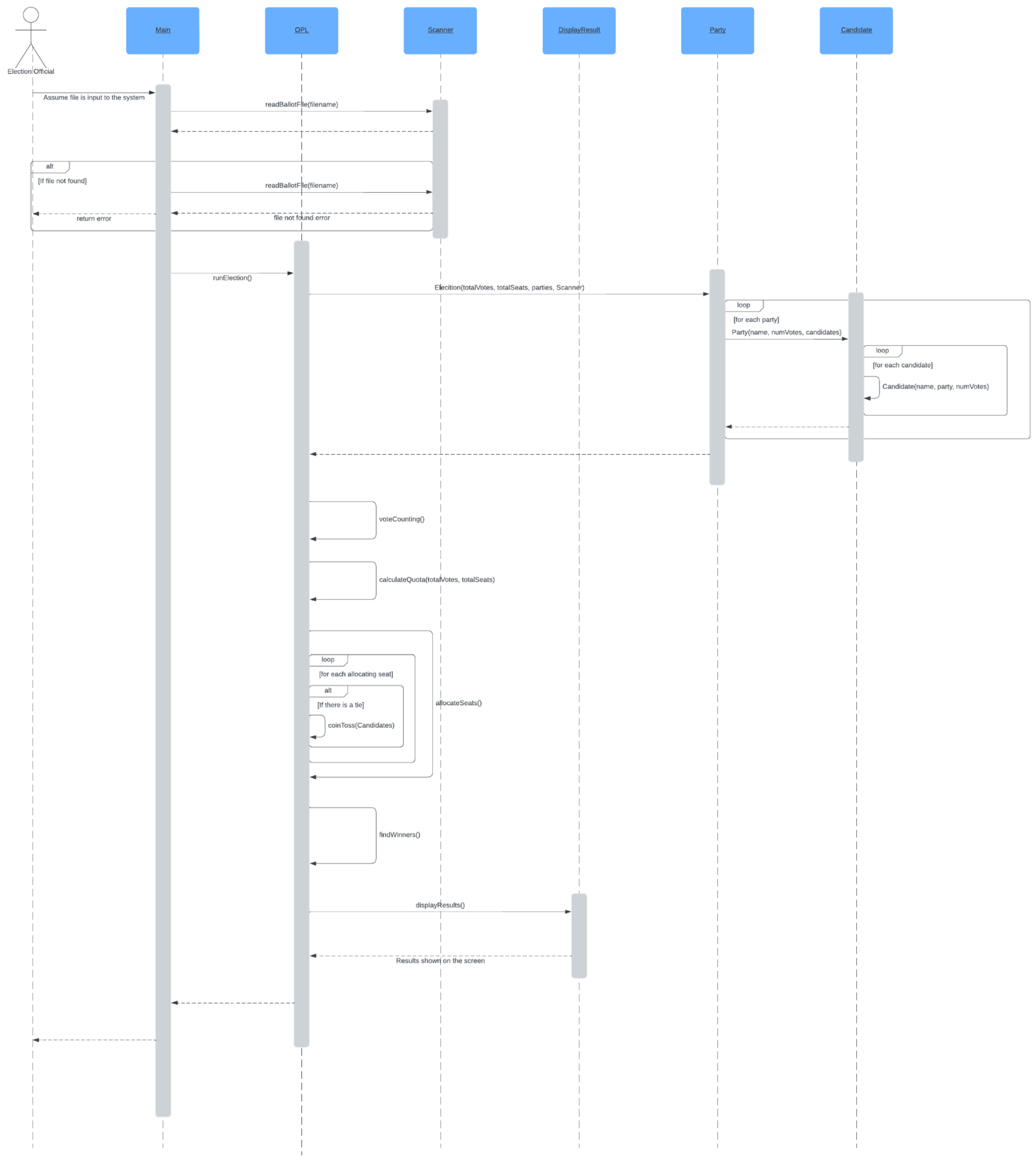


**Figure 3.2.2 Sequence Diagram for the running of the OPL election**

### 3.3 Design Rationale

While the file can be scanned all at once, the given CSV file is read in both readBallotFile() and voteCounting() in our system design because we want the system to count the votes for candidates or parties after instantiating Election, Party, and Candidate class. It would be more helpful to organize the data with the object-oriented system. Additionally, we originally created the calculateStats() method in the DisplayResults class to calculate the statistics such as the percentage of votes received. However, we removed it in the final design because it is a very simple calculation, and the input of this method is different between CPL and OPL. More specifically, the system should calculate the stats for parties in CPL, while it is for candidates in OPL.

## 4. Data Design

### 4.1 Data Description

The input and output files are retrieved and stored in the CSE lab machines at the University. The format of this input file is CSV. There are two types of choices: OPL and CPL. These two are also judged from the CSV file as well. The CSV file contains the data on the number of ballots, seats, and votes, candidate names, and their party names. This information will be stored in our Election class after judging the types of the election. The class will process and produce the necessary information: seat allocation, the winner, an audit file, and the election results. The output file (i.e. audit file) will be a text file.

### 4.2 Data Dictionary

**Audit Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| electionType | private | String | Two election types: OPL & CPL |
| numBallots | private | int | The number of ballots |
| numParties | private | int | The number of parties |
| numSeats | private | int | The number of seats |

| | | | |
|---|---|---|---|
| parties | private | ArrayList<Party> | A list of all the parties in this election |
| quota | private | int | Quota value used in the seat allocation algorithm |
| winnerList | private | ArrayList<Candidate> | List of the winners in the election |

| **Method:** | **Return type** | **Parameters** | **Description** |
|---|---|---|---|
| Audit | void | String, int, int, int, ArrayList<Candidate> | Creates an audit file based on the attributes |

**Candidate Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| name | private | String | Defining the names of the candidates |
| numVotes | private | int | Defining the number of votes |
| party | private | String | Defining the party name |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| Candidate | void | String, String, int | Constructor of the Candidate class |
| getName | String | - | Getting the names of the candidate |
| getNumVotes | int | - | Getting the number of votes |

| | | | |
|---|---|---|---|
| getParty | String | - | Getting the names of the party |
| setNumVotes | void | int | Setting the number of votes |

**CPL Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| fileReader | private | Scanner | Used to read the file |
| parties | private | ArrayList<Party> | Defining the parties |
| quota | private | int | This will be used in allocateSeat |
| totalSeats | private | int | Defining the total number of seats |
| totalVotes | private | int | Defining the number of votes |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| CPL | void | int, int, ArrayList<Party>, Scanner | Constructor for the CPL class |

**DisplayResults Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| electionType | private | String | Displaying OPL or CPL |
| numBallots | private | int | The number of |

| | | | |
|---|---|---|---|
| | | | ballots |
| numParties | private | int | The number of parties |
| numSeats | private | int | The number of seats |
| quota | private | int | The quotient between the total number of votes and the number of seats. Used to allocate seats |
| winnerList | private | ArrayList<Candidate> | Listing the winner in the election |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| DisplayResults | void | String, int, int, int, ArrayList<Candidate> | Displaying the election results based on the attributes |

**Election Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| fileReader | protected | Scanner | Used to read the file |
| totalSeats | protected | int | Defining the total number of seats |
| totalVotes | protected | int | Defining the total number of votes |
| parties | protected | ArrayList<Party> | Defining the parties |
| quota | protected | int | Setting what the quota is |

| | | | |
|---|---|---|---|
| winnerList | protected | ArrayList<Candidate> | Listing the winner in the election |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| allocateSeats | void | - | Allocate seats according to the Largest Remainder method |
| auditFile | void | - | Creating an output file (i.e. audit file) |
| calculateQuota | int | int, int | Calculated by total # of votes / total # of seats |
| coinToss | Candidate | ArrayList<Candidate> | This is a fair tiebreaker if two or more parties have the same # of votes |
| displayResult | void | - | Displaying the result on the users' screen based on the objects |
| Election | void | int, int, ArrayList<Party>, Scanner | Constructor of Election object |
| findWinners | void | - | Finds the winner of the election |
| voteCounting | void | - | Count the number of votes given to each party |

**Main Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| election | public | Election | Creating an object from the Election class |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| readBallotFile | void | String | Reading the given CSV files when running |
| runElection | void | - | Dealing with both of OPL and CPL, this method runs wholly |
| main | void | - | Complies the program |

**OPL Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| fileReader | private | Scanner | Reading a file containing necessary info |
| parties | private | ArrayList<Party> | Defining the parties |
| quota | private | int | This will be used in allocateSeat |
| totalSeats | private | int | Defining the total number of seats |
| totalVotes | private | int | Defining the total number of votes |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| auditFile | void | - | Creating an audit file |

| | | | |
|---|---|---|---|
| coinToss | Candidate | ArrayList<Candidate> | Uses a fair coin toss to break a tie between two or more candidates. |
| findWinners | void | - | Obtaining the winner based on the implementation |
| OPL | void | int, int, ArrayList<Party>, Scanner | Constructor of the OPL class |
| voteCounting | void | - | Counting the number of votes given to each candidate and party |

**Party Class:**

| **Attribute:** | **Attribute Accessibility:** | **Attribute Type:** | **Description:** |
|---|---|---|---|
| candidates | private | ArrayList<Candidate> | List of the candidates that belong to the party |
| name | private | String | Defining the names of the candidates in the party |
| numAllocatedSeats | private | int | Defining the number of allocated seats |
| numVotes | private | int | Defining the number of votes |

| **Method:** | **Return type:** | **Parameters:** | **Description:** |
|---|---|---|---|
| getCandidates | ArrayList<Candidate> | - | Obtaining the candidates' names |

| | | | |
|---|---|---|---|
| getName | String | - | Obtaining the name of the parties |
| getNumVotes | int | - | Obtaining the number of votes |
| Party | void | String, int, ArrayList<Candidate> | Constructor of the Party class |
| setNumAllocatedSeat | void | int | Setting the number of the allocated seat |
| setNumVotes | void | int | Setting the number of votes |

# 5. Component Design

## 5.1 Main

- **main()**

  We will run the program in this method.

- **readBallotFile (String filename)**

  This method will proceed to read the first line of the CSV file and assign this information to a class attribute called **electionType**.

- **runElection(): void**

  This method will instantiate and call the necessary methods to run CPL or OPL depending on the variable **electionType**. The necessary functions include:

  1. Call **voteCounting**() to get the number of votes each party gets.
  2. Call **calculateQuota()** to calculate the quota value.
  3. Call **allocateSeats()** to determine the number of seats each party gets.
  4. Call **findWinners()** to determine the winning candidate/party of the election.
  5. Call auditFile() and instantiate an **Audit** class to create an audit file.
  6. Call displayResults() and instantiate a **DisplayResults** class to display the results of the election.

## 5.2 Election

- **allocateSeats(): void**

  This method will calculate the number of seats each party wins using the largest remainder method. The procedure of calculating is as follows:
  1. Divide the number of votes that each party gets by the quota.
  2. Distribute remaining seats by comparing the largest remainder of votes for each party in a round-robin fashion.
- **voteCounting(): void**

  This method will count the number of ballots that each party gets from the CSV file.
- **coinToss(ArrayList<Party>): Party**

  This method will be used to break any ties throughout the program. It will take the array of Party and return a party representing the winning. The coin toss will run 1000 times and the last run, which is 1001, will be the result to ensure the coin is fair.
- **calculateQuota(int, int): int**

  The method then takes the quotient between the total number of ballots and the total number of seats to get the quota.
- **findWinners(): void**

  This method will return the party with the most seats and the candidate that is ranked first.
- **displayResults(): void**

  This method will instantiate the **DisplayResults** class.
- **auditFile(): void**

  This method will instantiate the **Audit** class.

## 5.3 CPL

- **allocateSeats(): void**

  This method will be inherited from the **Election** Class.
- **findWinners(): void**

  This method will be inherited from the **Election** Class.
- **voteCounting(): void**

  This method will be inherited from the **Election** Class
- **coinToss(ArrayList<Party>): Party for CPL**

  This method will be inherited from the **Election** Class.

- **displayResults(): void**

  This method will be inherited from the **Election** Class.

- **auditFile(): void**

  This method will be inherited from the **Election** Class.

## 5.4 OPL

- **allocateSeats(): void**

  This method will inherit from the **Election** Class.

- **findWinners(): void**

  This method will return the Candidate with the most votes.

- **voteCounting(): void**

  This method will count each line of the ballot file after it reads how many candidates it has. Each time it reads the ballot, it will increment the **count** of candidates that got the vote and the parties.

- **coinToss(ArryList<Candidates>): Candidate**

  This method will inherit from the **Election** class.

- **displayResults(): void**

  This method will instantiate the **DisplayResults** class.

- **auditFile(): void**

  This method will be inherited from the **Election** Class.

## 5.5 Party

- **getName(): String**

  This method will return the name of the party.

- **getNumVotes(): int**

  This method will return the total number of votes the party gets.

- **getCandidates(): ArrayList <Candidate>**

  This method will return an array of candidates that belong to the party.

- **setNumAllocatedSeats(): void**

  This method will set the number of seats each party gets.

- **setNumVotes(): void**

This method will set the value of numVotes in this object member variable.

## 5.6 Candidate

- **getName(): String**

  This method will return the name of the candidate.

- **getParty(): String**

  This method will return what party the candidate belongs to.

- **getNumVotes(): int**

  This method will return the number of votes the candidate gets.

- **setNumVotes(): void**

  This method will set the value of numVotes in this object member variable.

## 5.7 DisplayResults

- **displayResults(String, int, int, int): void**

  This method will show the overall results for the election. The results include the following information.

  1. Types of election (OPL or CPL).
  2. Number of parties.
  3. Number of seats.
  4. Number of ballots.
  5. The number of votes received and the percentage of votes received for each candidate in OPL and each party in CPL.
  6. A list of seat winners and their party affiliations.

## 5.8 Audit

- **audit(String, int, int, int): void**

  This method generates an audit file based on the election results. The file saves the following information.

  1. Type of election (OPL or CPL).
  2. Number of parties.
  3. Number of ballots.
  4. Number of seats.
  5. The candidates’ names and party affiliation.

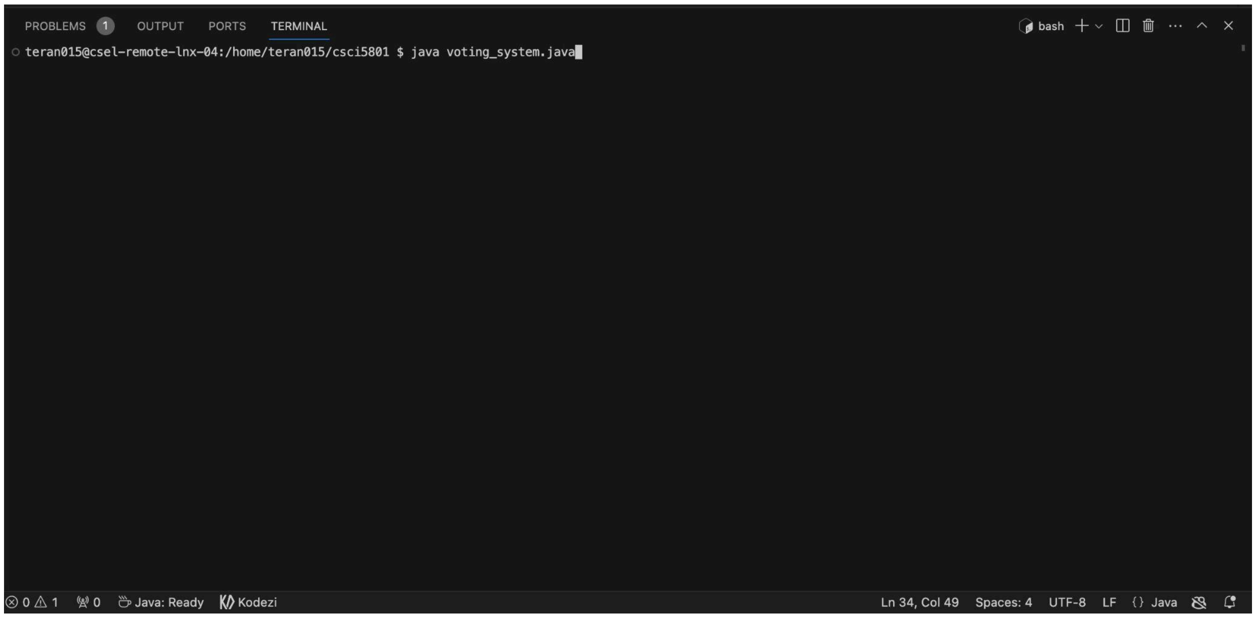
6. Quota value.
7. A list of seat winners and their party affiliations.

## 6. Human Interface Design

### 6.1 Overview of User Interface

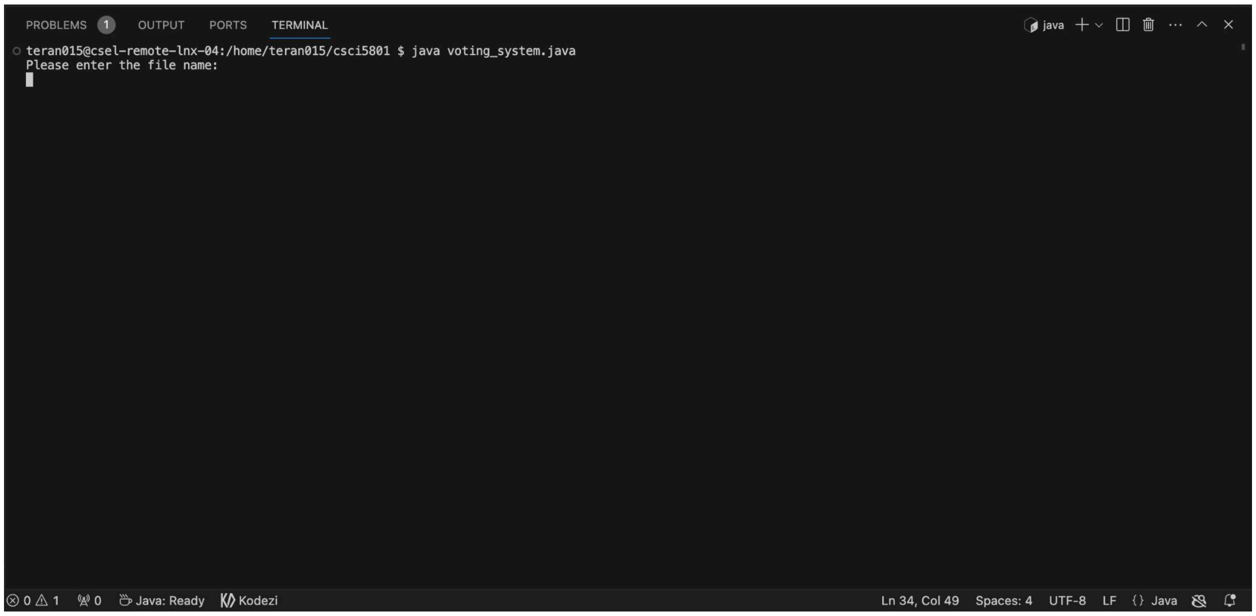
At the beginning, the user should enter a filename to access a given CSV file through the text prompt in the terminal. After reading the file, the system gets the necessary information: the election type, the party names, the candidate names, the ballots, and the number of seats, ballots, and parties or candidates. Next, the system will count the votes according to their election type. The next step is to allocate seats based on the Largest Remainder method. If there is a tie, a coin toss will randomly determine the winner. Lastly, the winner of the election will be determined and displayed along with election information. The system will also provide the user with an audit file containing the election results and information.

### 6.2 Screen Images



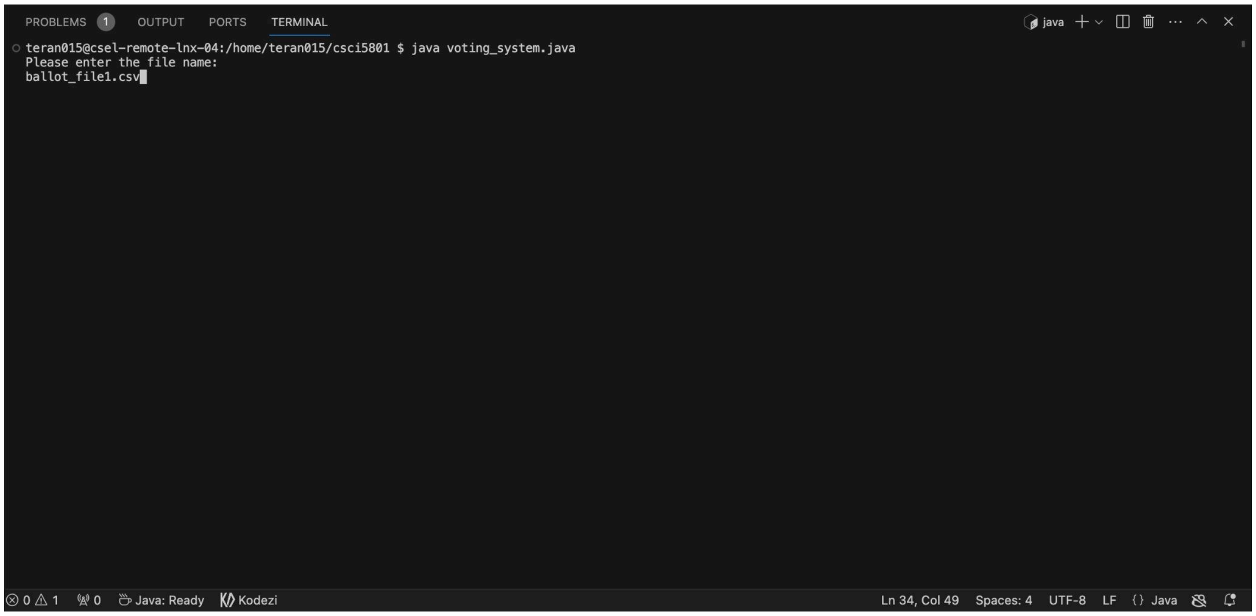
```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
```

*Figure 6.2.1: Set-up*



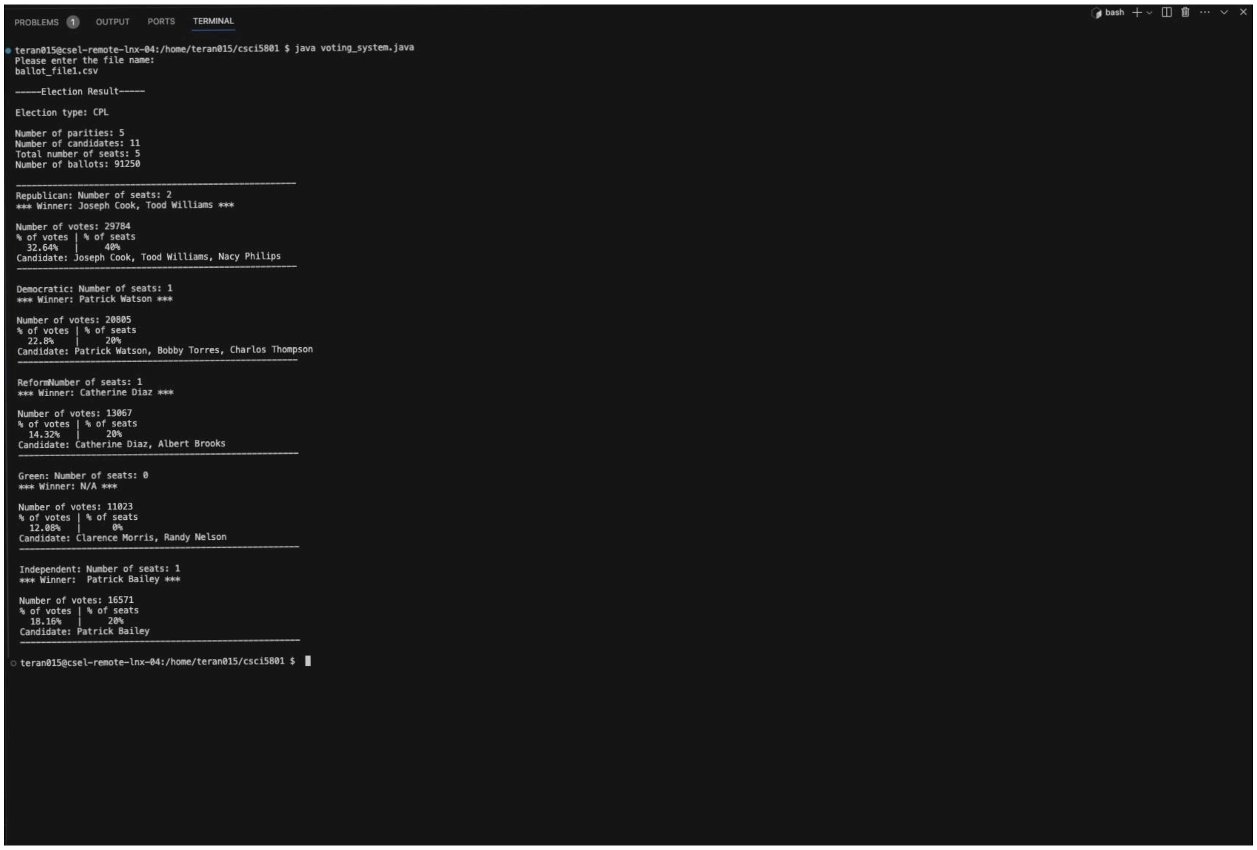
```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
Please enter the file name:
```

*Figure 6.2.2: After running the system*



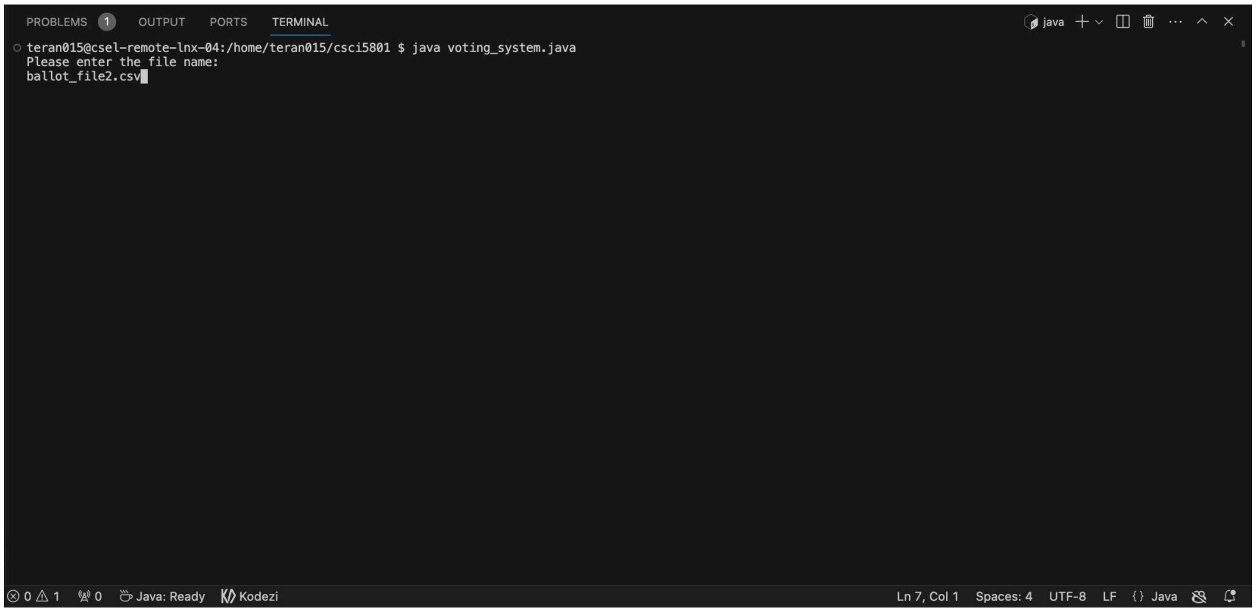
```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
Please enter the file name:
ballot_file1.csv
```

*Figure 6.2.3: Entering a filename*



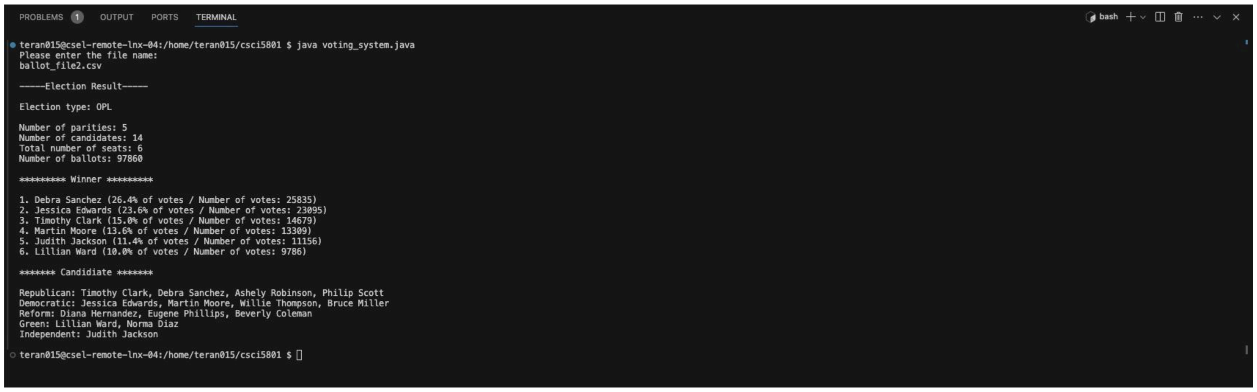
```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
Please enter the file name:
ballot_file1.csv

-----Election Result-----

Election type: CPL

Number of parities: 5
Number of candidates: 11
Total number of seats: 5
Number of ballots: 91250

------------------------------------------------------------
Republican: Number of seats: 2
*** Winner: Joseph Cook, Tood Williams ***

Number of votes: 29784
% of votes | % of seats
  32.64%   |     40%
Candidate: Joseph Cook, Tood Williams, Nacy Philips
------------------------------------------------------------

Democratic: Number of seats: 1
*** Winner: Patrick Watson ***

Number of votes: 20805
% of votes | % of seats
  22.8%    |     20%
Candidate: Patrick Watson, Bobby Torres, Charlos Thompson
------------------------------------------------------------

ReformNumber of seats: 1
*** Winner: Catherine Diaz ***

Number of votes: 13067
% of votes | % of seats
  14.32%   |     20%
Candidate: Catherine Diaz, Albert Brooks
------------------------------------------------------------

Green: Number of seats: 0
*** Winner: N/A ***

Number of votes: 11023
% of votes | % of seats
  12.08%   |      0%
Candidate: Clarence Morris, Randy Nelson
------------------------------------------------------------

Independent: Number of seats: 1
*** Winner:  Patrick Bailey ***

Number of votes: 16571
% of votes | % of seats
  18.16%   |     20%
Candidate: Patrick Bailey
------------------------------------------------------------

teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ 
```

*Figure 6.2.4: Election result in CPL*



```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
Please enter the file name:
ballot_file2.csv
```

*Figure 6.2.5: Entering a filename*



```
teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ java voting_system.java
Please enter the file name:
ballot_file2.csv

-----Election Result-----

Election type: OPL

Number of parities: 5
Number of candidates: 14
Total number of seats: 6
Number of ballots: 97860

********* Winner *********

1. Debra Sanchez (26.4% of votes / Number of votes: 25835)
2. Jessica Edwards (23.6% of votes / Number of votes: 23095)
3. Timothy Clark (15.0% of votes / Number of votes: 14679)
4. Martin Moore (13.6% of votes / Number of votes: 13309)
5. Judith Jackson (11.4% of votes / Number of votes: 11156)
6. Lillian Ward (10.0% of votes / Number of votes: 9786)

******* Candidiate *******

Republican: Timothy Clark, Debra Sanchez, Ashely Robinson, Philip Scott
Democratic: Jessica Edwards, Martin Moore, Willie Thompson, Bruce Miller
Reform: Diana Hernandez, Eugene Phillips, Beverly Coleman
Green: Lillian Ward, Norma Diaz
Independent: Judith Jackson

teran015@csel-remote-lnx-04:/home/teran015/csci5801 $ ▯
```

*Figure 6.2.6: Election result in OPL*

### 6.3 Screen Objects and Actions

There will be no screen objects. These screenshots show you the steps of how the system runs after getting the necessary information from the users as input. Then the images describe the election result after processing the inputs.

# 7. Requirements Matrix

| System Component: | Use Case in SRS: |
|---|---|
| Main: readBallotFile() | UC_001 |
| Main: runElection() | UC_002 |
| OPL | UC_003 |
| CPL | UC_004 |
| Election/OPL/CPL: allocateSeats() | UC_005 |
| Election: coinToss() | UC_006 |
| Election/OPL/CPL: findWinners() | UC_007 |
| Audit() | UC_008 |

| | |
|---|---|
| DisplayResults | UC_009 |

## 8. APPENDICES

There is no appendix for this document.